

REMARKS**Status of the Claims**

All pending claims 10-111 have been rejected.

Claim Rejections under 35 USC § 112

Claims 10-111 are rejected under 35 U.S.C. § 112, first paragraph, as failing to comply with the written description requirement.

Applicants' Response to Rejections under 35 USC § 112

The § 112, first paragraph rejection points to the last two steps of claim 10 as not being clearly disclosed in the original specifications or claims. Specifically, the Office action reads as follows:

The applicant claims that his system provides for “determining if a particular object-oriented method to be invoked during runtime execution is not present in executable program memory in the computer hardware and loading the particular object-oriented method into the executable program memory determined to not be present in the executable program memory prior to its runtime execution”; however, it is not clear that any of those features exists in the original specifications or claims.

In response, the Applicants would like to draw the Examiner's attention to the sections of the original specification that disclose features recited in the “determining” step and the “loading” of claim 10. For the convenience of the Examiner, citations are made to both the original great grand parent application S/N 08/094,675, filed on July 19, 1993 and to the published application 2004/0103416. The paragraph numbers of the published application are given in the style “[00xx]”. The text of the published application is identical to the text of the original great grand parent application S/N 08/094,675, filed on July 19, 1993, which was filed as the source document specification for the instant application.

A summary of the Applicants' overall response showing that there is sufficient disclosure in the application, is provided on page 12, line 25 to page 13, line 3 in Original application S/N 08/094,675, July 19, 1993, which is Paragraph [0055] in the published application 2004/0103416. The paragraph reads as follows:

Referring now to FIG. 3, in step 308, **it is determined whether the computer program logic (also called computer code) from the code library 110 which implements the method referenced in the statement is present in the task address space associated**

with the application 130A. If the computer program logic is present in the task address space, then step 316 is processed (described below). **If the computer program logic is not present in the task address space, then the computer program logic is transferred to the task address space in steps 310, 312, and 314.** In step 310, it is determined whether the library server (not shown) associated with the code library 110 is known. The code library 110 may represent multiple code libraries (not shown) related to the wrapper 128, wherein each of the code libraries include the computer program logic for one of the object-oriented classes of the class library 402. As those skilled in the relevant art will appreciate, there may also be other code libraries (not shown) completely unrelated to the wrapper 128.

The above quotation refers to Figure 3, which appears in the application as follows:

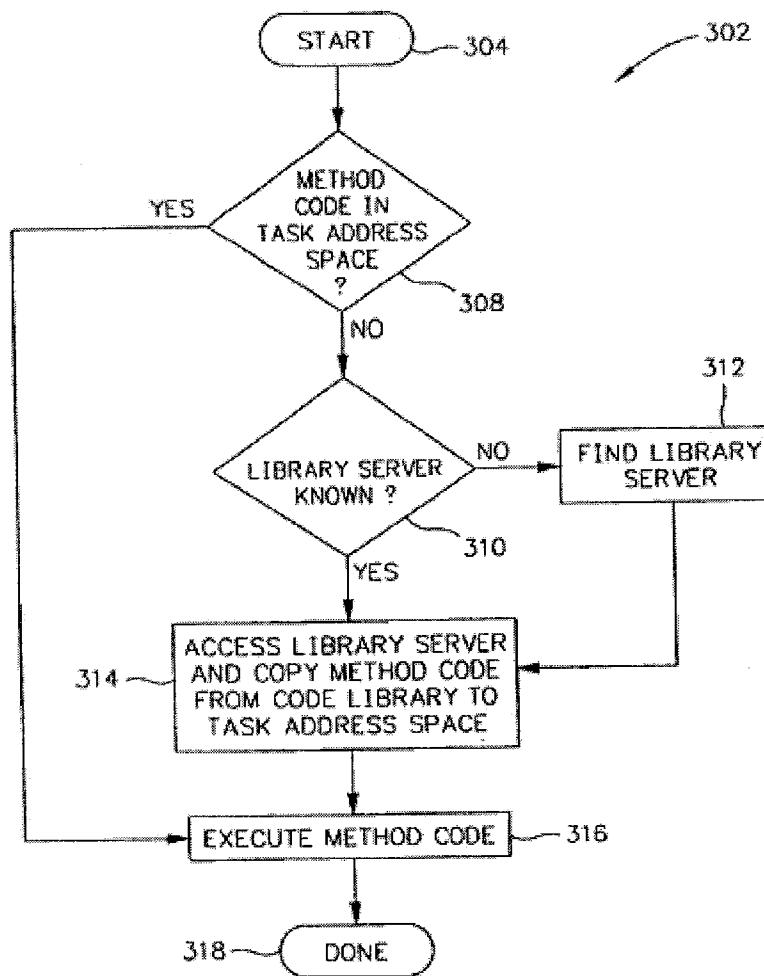


FIG. 3

The term “**determining if a particular object-oriented method ... is not present**” corresponds to step 308 in Figure 3, which is the decision step “Method code in task address space?” and has a “No” result passing to the next step. The “particular object-oriented method” is the wrapper-method and the code that implements the wrapper-method, as described in paragraphs [0055], [0042], [0040], and [0058], which read as follows:

[Page 12, line 25-32 in Original application S/N 08/094,675, July 19, 1993] / Paragraph [0055]

Referring now to FIG. 3, in step 308, it is determined whether the computer program logic (also called computer code) from the code library 110 which implements the method referenced in the statement is present in the task address space associated with the application 130A. If the computer program logic is present in the task address space, then step 316 is processed (described below). If the computer program logic is not present in the task address space, then the computer program logic is transferred to the task address space in steps 310, 312, and 314. In step 310, it is determined whether the library server (not shown) associated with the code library 110 is known.

[Page 9, line 16-23 in Original application S/N 08/094,675, July 19, 1993] / Paragraph [0040]

The wrapper 128 is preferably implemented as a code library 110 ... The code library 110 implements an object-oriented class library 402 (see FIG. 4). ... The object-oriented classes comprise methods

[Page 10, line 12-14 in Original application S/N 08/094,675, July 19, 1993] / Paragraph [0042]

...The **class library 402** of the present invention includes a set of related **classes for each of the Mach service categories**. Referring to FIG. 4, the class library 402 includes

[Page 12, line 5-8 in Original application S/N 08/094,675, July 19, 1993] / Paragraph [0054]

The operation of a preferred embodiment shall now be described in more detail with reference to FIG. 3, which illustrates a detailed operational flow chart 302 of the present invention. Again, the present invention is described in the context of **executing the object-oriented application 130A** on the computer platform 102.

[Page 14, line 12-16 in Original application S/N 08/094,675, July 19, 1993] / Paragraph [0058]

... For example, the run-time conventions may specify that when an instruction accessing an operating system service is encountered, corresponding **code from the code library 110** should be **transferred to the task address space** (via the associated library server) **and executed**.

The term “**executable program memory**” is a task address space, as described in paragraphs [0054], [0055], [0058], [0063], [0065], and [0067], which read as follows:

[Page 12, line 5-24 in Original application S/N 08/094,675, July 19, 1993] / Paragraph [0054]

The operation of a preferred embodiment shall now be described in more detail with reference to FIG. 3, which illustrates a detailed operational flow chart 302 of the present invention. ... The application 130A includes statements which access services provided by the operating system 114, and it is assumed that such statements are defined by the class library 402 (in other words, the programmer created the application 130A with reference to the class library 402). As will be discussed in greater detail below, the executable entity in the Mach micro-kernel is called a thread. The processing organization entity in the Mach micro-kernel is called a task. **A task includes one or more threads (which may execute in parallel), and an address space which represents a block of virtual memory in which the task's threads can execute.** ... **When executing on the computer platform 102, the application 130A could represent an entire task** (having one or more threads), or could represent a few threads which are part of a task (in this case, the task would have other threads which may or may not be related to the operation of the application 130A). The scope of the present invention encompasses the case when the application 130A is an entire task, or just a few threads of a task.

[Page 12, line 25-28 in Original application S/N 08/094,675, July 19, 1993] / Paragraph [0055]

Referring now to FIG. 3, in step 308, it is determined whether the computer program logic (also called computer code) from the code library 110 which implements **the method referenced in the statement is present in the task address space associated with the application 130A.**

[Page 14, line 13-16 in Original application S/N 08/094,675, July 19, 1993] / Paragraph [0058]

For example, the run-time conventions may specify that when an instruction accessing an operating system service is encountered, **corresponding code from the code library 110 should be transferred to the task address space (via the associated library server) and executed.**

[Page 17, line 5-6 in Original application S/N 08/094,675, July 19, 1993] / Paragraph [0063]

... A task also has an address space, which represents virtual memory in which its threads can execute.

[Page 18, line 28 to page 19, line 2 in Original application S/N 08/094,675, July 19, 1993] / Paragraph [0065]

... The address space of a Mach **task contains a number of virtual memory regions**. These **regions are pieces of virtual address space** that have been allocated in various ways **for use by the task**. They are the only locations **where memory can be legitimately accessed**. All references to addresses outside of the defined regions in the address space will result in an improper memory reference fault. A virtual memory region has several interesting attributes. It has a page-aligned starting address and a size, which must be a multiple of the system page size. The pages in the region all have the same **access protections**; these access protections can be read-only, read-write, or **execute**.

[Page 20, line 12-14 in Original application S/N 08/094,675, July 19, 1993] / Paragraph [0067]

Most clients, however, will deal with virtual memory through operations on **ranges of memory**. A range can be a portion of a region or it could span multiple contiguous regions **in the address space**.

The term “**invoked**” is to cause the code associated with a method to be executed and corresponds to step 316 of Fig. 3 that says “Execute Method Code”, and which is described in paragraphs [0005] and [0057], which read as follows:

[Page 2, line 1-5 in Original application S/N 08/094,675, July 19, 1993] / Paragraph [0005]

A **method in an object is invoked by passing a message to the object** (this process is called message passing). The message specifies a method name and an argument list. **When the object receives the message, code associated with the named method is executed** with the formal parameters of the method bound to the corresponding values in the argument list.

[Page 13, line 33 to page 14, line 1 in Original application S/N 08/094,675, July 19, 1993] / Paragraph [0057]

... In step 316, the computer program logic associated with the object-oriented statement is **executed** on the computer platform 102.

The term “**computer hardware**” is, *inter alia*, a random access memory (RAM) and a central processing unit, as described in paragraph [0036], which reads as follows:

[Page 7, line 16-31 in Original application S/N 08/094,675, July 19, 1993] / Paragraph [0036]

FIG. 1 illustrates a block diagram of a computer platform 102 in which a wrapper 128, 129 of the present invention operates. It should be noted that **the present invention alternatively encompasses the wrapper 128, 129 in combination with the computer**

platform 102. The computer platform 102 includes **hardware components 103, such as a random access memory (RAM) 108 and a central processing unit (CPU) 106.** It should be noted that the CPU 106 may represent a single processor, but preferably represents multiple processors operating in parallel. The computer platform 102 also includes peripheral devices which are connected to the hardware components 103. These peripheral devices include an input device or devices (such as a keyboard, a mouse, a light pen, etc.), a data storage device 120 (such as a hard disk or floppy disk), a display 124, and a printer 126. The data storage device 120 may interact with a removable data storage medium 122 (such as a removable hard disk, a magnetic tape cartridge, or a floppy disk), depending on the type of data storage device used. **The computer platform 102 also includes a procedural operating system 114 having a native procedural interface (not shown).** The procedural interface includes procedural functions which are called to access services provided by the operating system 102.

The term “**loading the particular object-oriented method**” is transferring or copying the wrapper-method and the code that implements the method into the task address space of the application, as described in paragraphs [0055], [0057] and [0040], which read as follows:

[Page 12, line 29 to page 13, line 3 in Original application S/N 08/094,675, July 19, 1993 / Paragraph [0055]]

... If the computer program logic is not present in the task address space, then the computer program logic is **transferred** to the task address space in steps 310, 312, and 314. In step 310, it is determined whether the library server (not shown) associated with the code library 110 is known. The code library 110 may represent multiple code libraries (not shown) related to the wrapper 128, wherein each of the code libraries include the computer program logic for one of the object-oriented classes of the class library 402. As those skilled in the relevant art will appreciate, there may also be other code libraries (not shown) completely unrelated to the wrapper 128.

[Page 13, line 22 to page 14, line 1 in Original application S/N 08/094,675, July 19, 1993] / Paragraph [0057]

... **In step 314, a request is sent to the library server asking the library server to copy the computer program logic associated with the method reference in the statement to the task address space. Upon completion of step 314, the library server has copied the requested computer program logic to the task address space.** ... Preferably, the code library 110 is a shared library. That is, the code library 110 may be simultaneously accessed by multiple threads. However, preferably the computer program logic of the code library 110 is physically stored in only one physical memory area. The library server **virtually copies** computer program logic from the code library 110 to task address spaces. That is, instead of **physically copying** computer program logic from one part of physical memory to another, the library server places in the task address space a

pointer to the physical memory area containing the relevant computer program logic. In step 316, the computer program logic associated with the object-oriented statement is executed on the computer platform 102.

[Page 9, line 16-23 in Original application S/N 08/094,675, July 19, 1993] / Paragraph [0040]

The wrapper 128 is preferably implemented as a code library 110 ... The code library 110 implements an object-oriented class library 402 (see FIG. 4). ... The object-oriented classes comprise methods

The Applicants believe that their claimed invention is adequately disclosed in their specification and figures, as originally filed in their original great grand parent application S/N 08/094,675, filed on July 19, 1993, and fully complies with under 35 USC § 112, paragraph 1.

Claim Rejections under 35 USC § 102

Claims 10-111 are rejected under 35 U.S.C. § 102(e) as being anticipated by Nakano et al. (US 5,369,766).

Applicants' Response to Rejections under 35 USC § 102

Specifically, the Office action provides a claim table that reads terms in the steps of claim 10 on Nakano et al. (US 5,369,766).

For the first element, “providing an object-oriented interface,” in the Applicants’ claim 10, the Office action claim table reads the claim term:

“the interface implemented on a plurality of computer platforms including different combinations of computer hardware and operating systems,”

on Nakano’s column 11, lines 25-35 and Nakano’s claim 1. The cited sections of Nakano read as follows:

[Nakano column 11, lines 25-35]:

Within the **same CPU architecture**, load modules created by **different languages** and by **different development environments** mix-and-match. The preferred embodiment of the preferred embodiment requires that addresses corresponding to exported functions and static data items follow a specific CPU's calling convention, which may involve providing addresses that correspond to

translating between the preferred embodiment of the subject preferred embodiment's calling conventions, and a different internal set of conventions.

[Nakano claim 1]:

1. An apparatus for processing a plurality of load modules in **a computer**, said plurality of load modules created by a plurality of development environments resulting in different load module formats, comprising:
 - (a) **the computer**;
 - (b) a storage in the computer;
 - (c) processing means in the computer for loading a first object-oriented load module of a first format in said address space;
 - (d) processing means in the computer for loading a second load module which can be non-object-oriented of a second format in said address space; and
 - (e) processing means in the computer for having said first object-oriented load module and said second load module execute in said address space and utilize portions of said second load module for execution of said first object-oriented load module.

The Applicants reply that Nakano fails to disclose or suggest the Applicants' claimed "interface implemented on a plurality of computer platforms including **different combinations of computer hardware and operating systems**." The Nakano disclosure at column 11 reads: "Within **the same CPU architecture**, load modules created by different languages and by different development environments mix-and-match." Nakano's column 11 discloses the same concept as is disclosed in Nakano's claim 1, which reads: "processing a plurality of load modules **in a computer**." This cited disclosure in Nakano describes running applications with different languages concurrently **in one computer**. There is no disclosure or suggestion of the Applicants' claimed "interface implemented on **a plurality of computer platforms including different combinations of computer hardware and operating systems**."

For the second element, "the program logic code," in the Applicants' claim 10, the Office action claim table reads the claim element:

"the program logic code responsive to the object-oriented interface to provide native system services from the computer platform"

on Nakano's column 2, line 65-column 3, line 11. The cited section of Nakano reads as follows:

[Nakano column 2, line 65-column 3, line 11]:

A load module implements functions, static data, and classes. TLoadModule, an abstract class, defines the protocol to load and access load modules. TLoadModule allows subclasses to use very different approaches to represent and implement functions, static data, and classes. For example, a ROM on a NuBus card, a boot partition on a disk, or a disk file, can all appear as load modules. Similarly, different programming languages and development environments can use pre-existing TLoadModule subclasses, or design custom subclasses, to suit special needs. The TLoadModule class provides a common protocol for the loader to mix and match fundamentally different kinds of containers of data built by diverse development environments.

The Applicants reply that Nakano fails to disclose or suggest the Applicants' claimed "the program logic code responsive to the object-oriented interface to provide **native system services** from the computer platform." The Nakano disclosure at column 2 reads: "A load module implements functions, static data, and classes." This cited disclosure in Nakano describes that a "load module" implements functions, static data, and classes, which are **ordinary portions of an application program**. But, **there is no disclosure or suggestion** of the Applicants' claimed program logic code to provide **native system services** from the computer platform. Native system services are services provided by an underlying operating system which are functionalities that **cannot otherwise be readily implemented in an application program** including services such as tasks, virtual memory, interprocess communications (IPC), synchronization, scheduling, fault, machine, and security services, as shown in Figure 4:

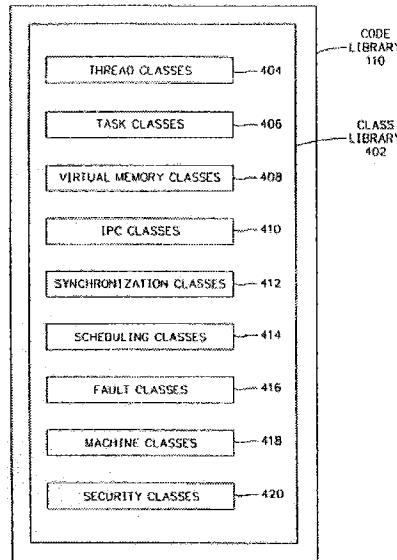


FIG. 4

Nowhere in Nakano does the term “services” appear, nor does any teaching or suggestion that the code loaded by Nakano is for enabling an application to access system services such as those enumerated in Fig. 4 that are provided by and native to the underlying computer platform (operating system and computer hardware) in the Applicants’ claimed invention.

Since the two claim terms discussed above for the Applicants’ claim 10, i.e. “interface implemented on a plurality of computer platforms” and “program logic code to provide native system services,” also appear in all of the Applicants’ remaining independent claims, Nakano fails to disclose or suggest the Applicants’ claimed invention, as discussed above, for all of Applicants’ claims 10-111.

CONCLUSION

Based on the foregoing remarks, Applicants respectfully request reconsideration and withdrawal of the rejection of claims and allowance of this application.

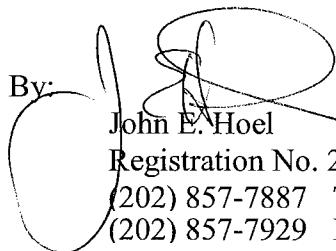
AUTHORIZATION

The Commissioner is hereby authorized to charge any additional fees which may be required for consideration of this Amendment to Deposit Account No. 13-4500, Order No. 4386-7004US1. A DUPLICATE OF THIS DOCUMENT IS ATTACHED.

In the event that an extension of time is required, or which may be required in addition to that requested in a petition for an extension of time, the Commissioner is requested to grant a petition for that extension of time which is required to make this response timely and is hereby authorized to charge any fee for such an extension of time or credit any overpayment for an extension of time to Deposit Account No 13-4500, Order No. 4386-7004US1. A DUPLICATE OF THIS DOCUMENT IS ATTACHED.

Respectfully submitted,
MORGAN & FINNEGAN, L.L.P.

Dated: June 7, 2007

By: 
John E. Hoel
Registration No. 26,279
(202) 857-7887 Telephone
(202) 857-7929 Facsimile

Correspondence Address:

MORGAN & FINNEGAN, L.L.P.
3 World Financial Center
New York, NY 10281-2101